Trees need care

a solution to Device Tree validation problem

April 30, 2014
Embedded Linux Conference
San Jose, CA

Tomasz Figa
Linux Kernel Developer
Samsung R&D Institute Poland

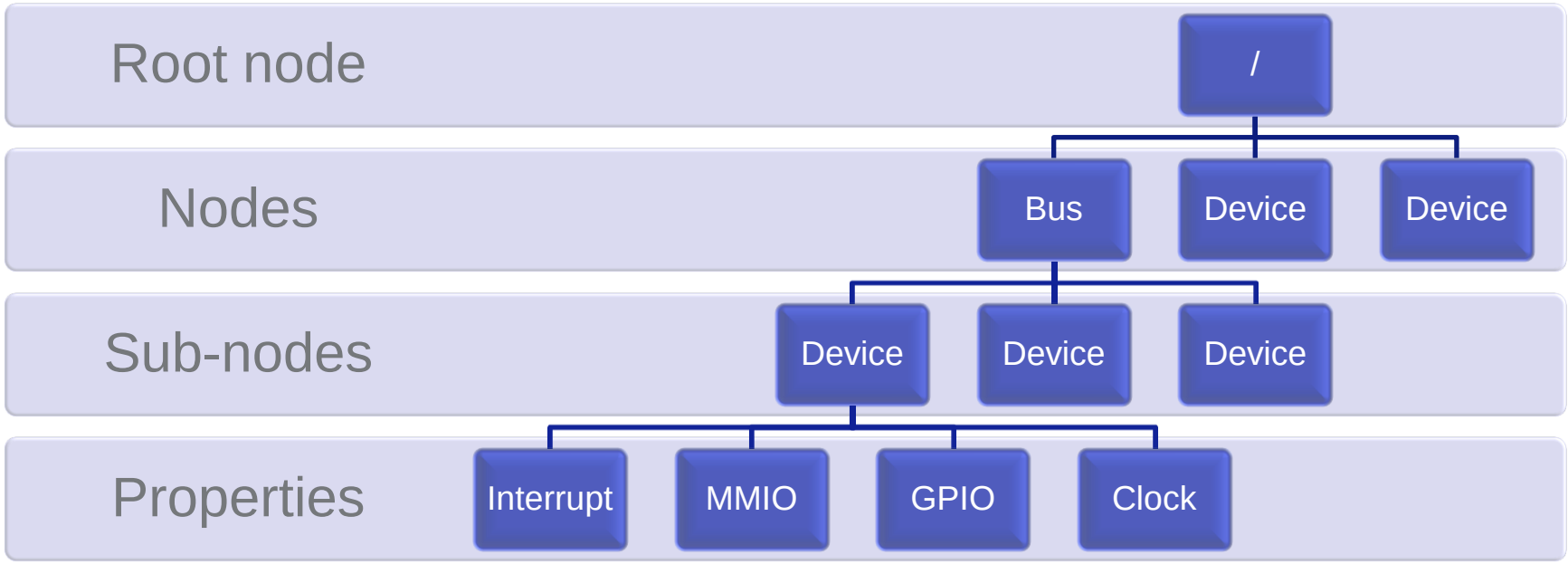
SAMSUNG

Overview

1. Device Tree recap
2. Device Tree data flow
3. What’s wrong?
4. Problem analysis
5. History of work
6. Solution proposal
7. What’s next?
8. Q&A

A quick recap.

Device Tree



Data structure for describing hardware

Needed to tell the OS about devices that cannot be detected automatically

Tree-like structure correllated with hardware topology

Describes resources needed by devices

Represents relations between devices

Passed to kernel at boot

Replaces hard-coded platform details in the OS

Terminology:

Property:

a key-value pair; key – property name, value – arbitrary data.

Node:

a set of properties and/or child nodes.

Bindings:

a description how a device is described using device tree; a list of properties and nodes (and their formats) necessary to describe a device.

Widely adopted

SPARC, PowerPC, ARM, MIPS, C6x, Metag, OpenRISC, Xtensa, Microblaze, ARC

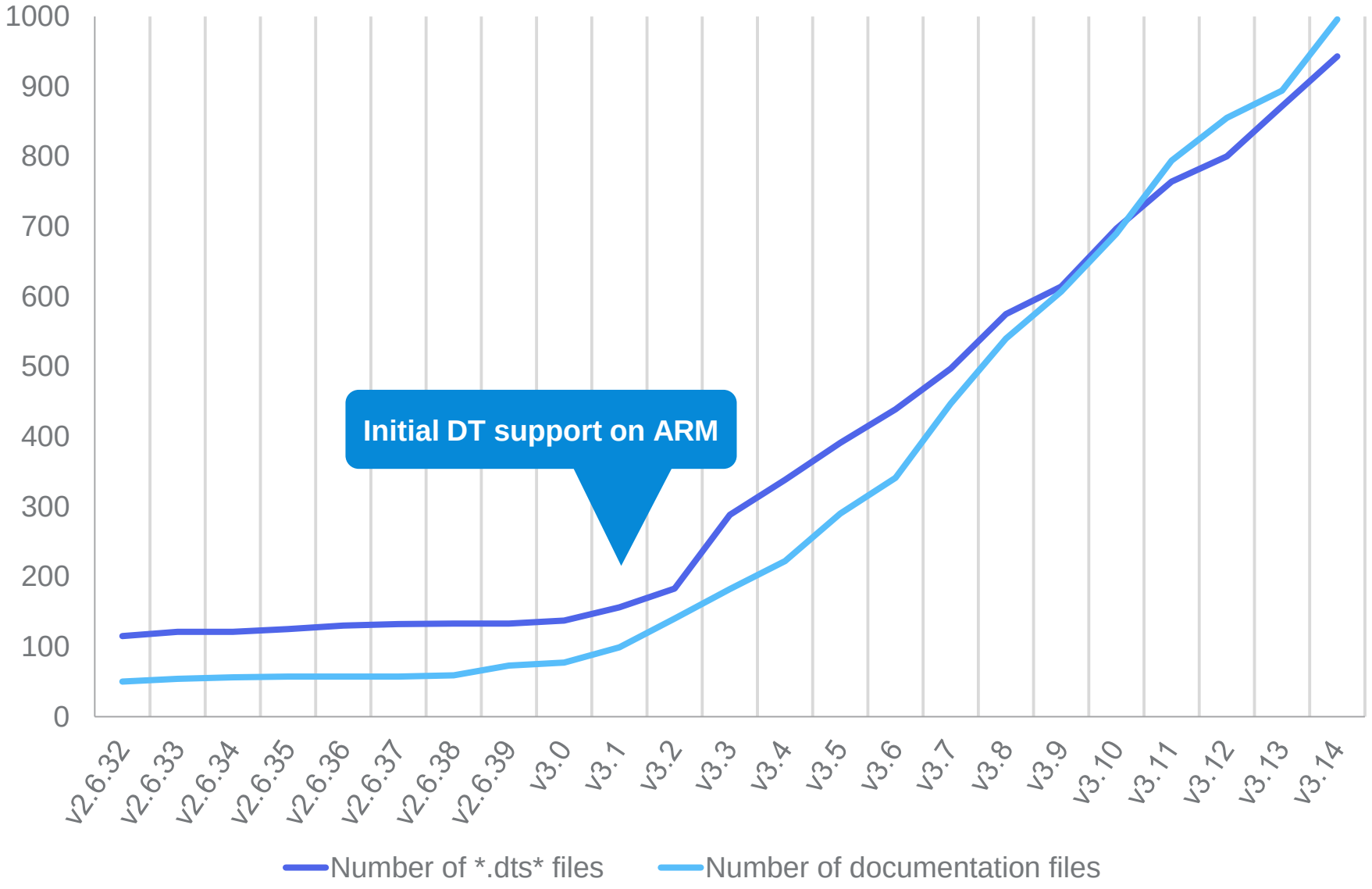
Even x86 ;-)

More than 600 in-tree board device tree source files*

More than 4000 compatible strings defined*

*As of Linux 3.14

Device Tree


1000
900
800
700
600
500
400
300
200
100
0
Initial DT support on ARM
v2.6.32
v2.6.33
v2.6.34
v2.6.35
v2.6.36
v2.6.37
v2.6.38
v2.6.39
v3.0
v3.1
v3.2
v3.3
v3.4
v3.5
v3.6
v3.7
v3.8
v3.9
v3.10
v3.11
v3.12
v3.13
v3.14
Number of *.dts* files
Number of documentation files

What's happening to our trees?

Device Tree data flow

Device Tree Source (DTS)

Plain text

Human-readable

Developer- and GIT-friendly

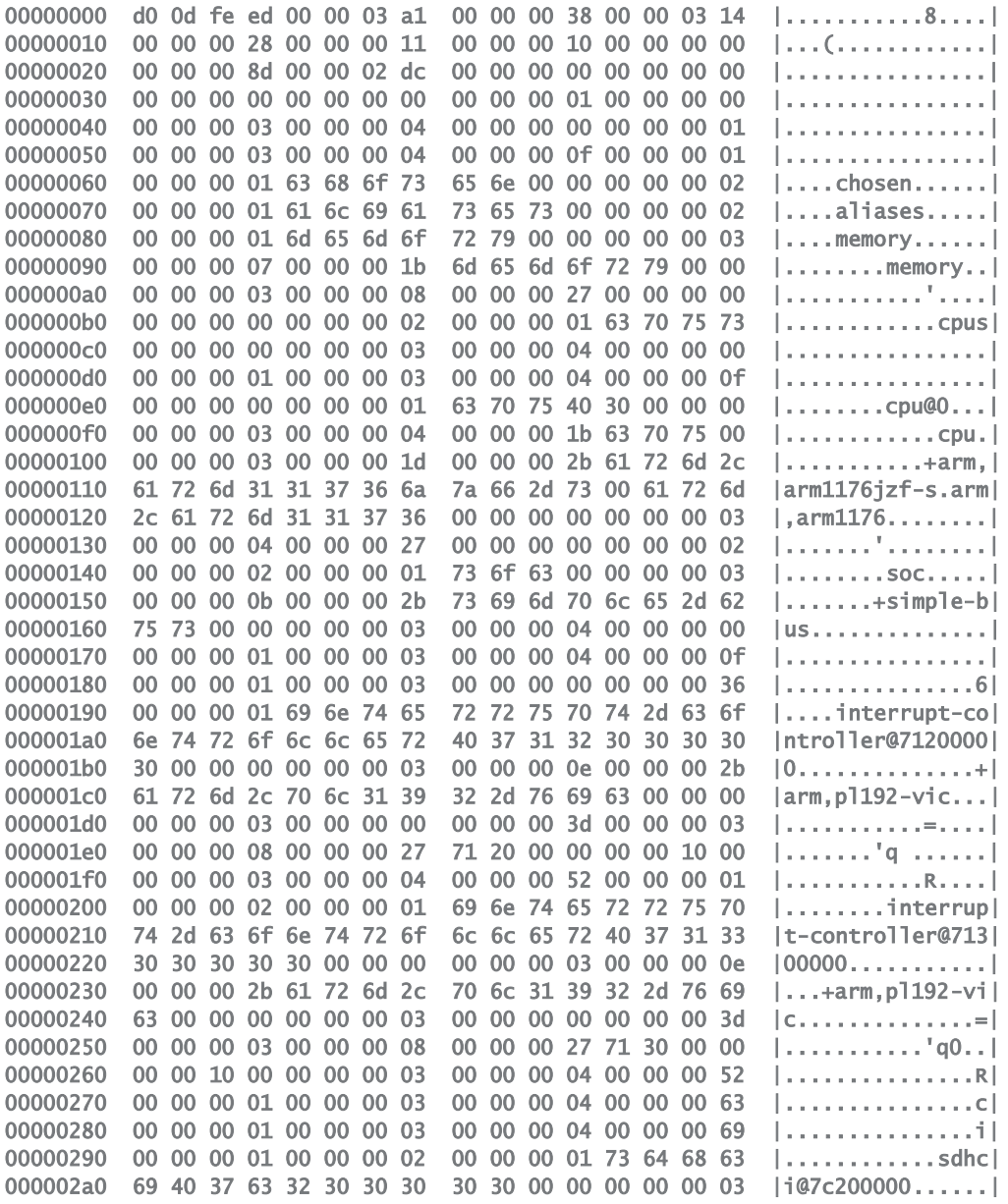
```
[...]
        cpus {
                #address-cells = <1>;
                #size-cells = <0>;

                cpu@0 {
                        device_type = "cpu";
                        compatible = "arm,arm1176jzf-s", "arm,arm1176";
                        reg = <0x0>;
                };
        };

        soc: soc {
                compatible = "simple-bus";
                #address-cells = <1>;
                #size-cells = <1>;
                ranges;

                vic0: interrupt-controller@71200000 {
                        compatible = "arm,pl192-vic";
                        interrupt-controller;
                        reg = <0x71200000 0x1000>;
                        #interrupt-cells = <1>;
                };

                vic1: interrupt-controller@71300000 {
                        compatible = "arm,pl192-vic";
                        interrupt-controller;
                        reg = <0x71300000 0x1000>;
                        #interrupt-cells = <1>;
                };

                sdhci0: sdhci@7c200000 {
                        compatible = "samsung,s3c6410-sdhci";
                        reg = <0x7c200000 0x100>;
                        interrupt-parent = <&vic1>;
                        interrupts = <24>;
                };
[...]
```

Device Tree Blob (DTB)

Binary representation called Flattened Device Tree (FDT)

Machine-readable

CPU-friendly



Device Tree Compiler (DTC) [1]

Translates between various DT representations

Usually DTS -> DTB

Contains simple DTS parser built with Flex+Bison

[1] DTC GIT repo at kernel.org

A story of a failure.

What’s wrong?

DTC translates data directly from one representation to another

Every node/property reaches output file

Almost no checks performed on input data

Let’s see...

What's wrong?

sample.dts (part of arch/arm/boot/dts/s3c64xx.dtsi)

```
[...]
            vic0: interrupt-controller@71200000 {
                  compatible = "arm,pl192-vic";
                  interrupt-controlller;
                  reg = <0x71200000 0x1000>;
                  #interrupt-cells = <1>;
            };

            vic1: interrupt-controller@71300000 {
                  compatible = "arm,pl192-vic";
                  interrupt-controller;
                  reg = <0x71300000 0x1000>;
                  #interrupt-calls = <1>;
            };

            sdhci0: sdhci@7c200000 {
                  compatible = "samsung,s3c6410-sdhci";
                  reg = <0x7c200000 0x100>;
                  interrupt-parent = <&vic1>;
                  interrupts = <24 0>;
[...]
```

How many errors can you spot in this snippet?

What's wrong?

sample.dts (part of arch/arm/boot/dts/s3c64xx.dtsi)

```
[...]
            vic0: interrupt-controller@71200000 {
                  compatible = "arm,pl192-vic";
                  interrupt-controller;
                  reg = <0x71200000 0x1000>;
                  #interrupt-cells = <1>;
            };

            vic1: interrupt-controller@71300000 {
                  compatible = "arm,pl192-vic";
                  interrupt-controller;
                  reg = <0x71300000 0x1000>;
                  #interrupt-calls = <1>;
            };

            sdhci0: sdhci@7c200000 {
                  compatible = "samsung,s3c6410-sdhci";
                  reg = <0x7c200000 0x100>;
                  interrupt-parent = <&vic1>;
                  interrupts = <24 0>;
[...]
```

And the answer is…

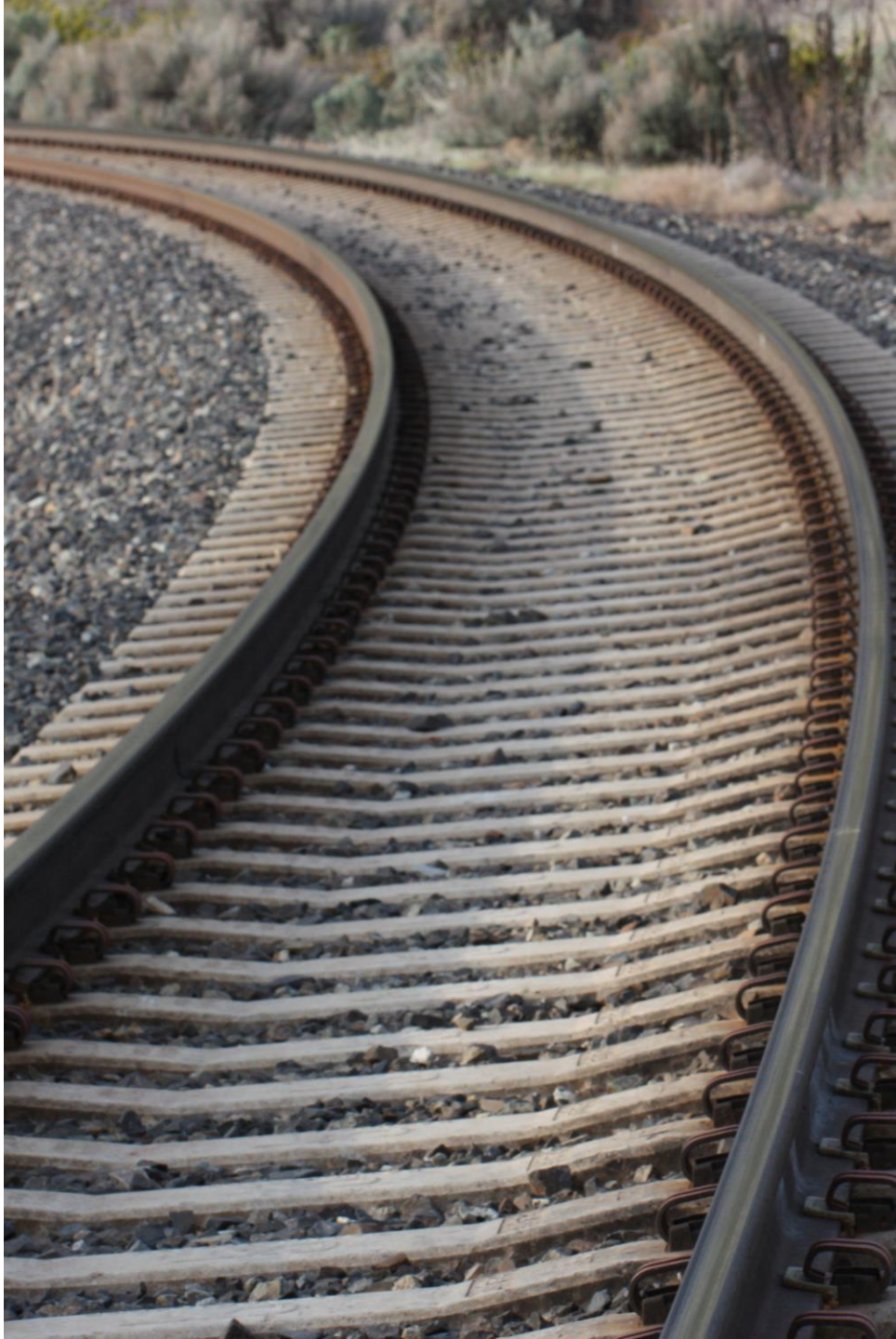
What's wrong?

```
t.figa@AMDC1227 ~/kernel $ scripts/dtc/dtc -O dtb -o sample.dtb arch/arm/boot/dts/sample.dts
t.figa@AMDC1227 ~/kernel $ ls -l sample.dtb
-rw-r--r-- 1 t.figa t.figa 929 04-21 14:21 sample.dtb
t.figa@AMDC1227 ~/kernel $
```

Hmm... Zero?

Getting back on the right track.

Device Tree validation



Bindings define nodes and their contents

Documented in a human-readable format (varying between files)

Non-parsable

Validation of DT sources done by IVM*

* Intelligent Validation Machines aka Maintainers

```
* ARM Vectored Interrupt Controller

One or more Vectored Interrupt Controllers (VIC's) can
system for interrupt routing.  For multiple controllers
nested or have the outputs wire-OR'd together.

Required properties:

- compatible : should be one of
»       "arm,pl190-vic"
»       "arm,pl192-vic"
- interrupt-controller : Identifies the node as an inte
- #interrupt-cells : The number of cells to define the
  the VIC has no configuration options for interrupt so
  and defines the interrupt number.
- reg : The register bank for the VIC.

Optional properties:

- interrupts : Interrupt source for parent controllers
- valid-mask : A one cell big bit mask of valid interru
  represents single interrupt source, starting from sou
  at source 31 at MSb. A bit that is set means that the
  clear means otherwise. If unspecified, defaults to al
- valid-wakeup-mask : A one cell big bit mask of interr
  configured as wake up source for the system. Order of
  valid-mask property. A set bit means that this interr
  configured as a wake up source for the system. If uns
  interrupt sources configurable as wake up sources.

Example:

»       vic0: interrupt-controller@60000 {
»       »       compatible = "arm,pl192-vic";
```

Conclusion 1

We need a formal way to describe DT bindings

In March 2012, Jon Smirl posted on devicetree-discuss@lists.ozlabs.org [1]:

„In the XML world you can make schemas that describe what is legally allowed in an XML document. Device trees don't seem to any any kind of schema mechanism describing what is and isn't legal to put in a tree.

Instead of recreating the entire world of schemas, has anyone looked at using XML schemas to define what is a valid device tree and then write a tool to validate the existing device trees?”

The topic did not receive much attention, though.

[1] Schemas for device trees

RFC by Benoit Cousson and Fabien Parent [1]

September 2013

DTS-like schema language

Simple, readable, nice

Not flexible enough

```
/dts-v1/;
/ {
    compatible = "ti,omap[0-9]+-timer";
    ti,hwmods {
        is-required;
    };
};
```

```
/dts-v1/;
/ {
    compatible = "ti,twl[0-9]+-rtc";
    interrupt-controller {
        is-required;
        can-be-inherited;
    };
};
```

```
/ {
    compatible = "abc";
    abc {
        type = "integer";
    };
    def {
        type = "string";
    };
};
```

[1] [RFC 00/15] Device Tree schemas and validation

RFC by Tomasz Figa [1]

October 2013

Another DTS-like schema language

Sooo complex, ugh…

Still not flexible enough…

```
/template/ interrupts {
    /arg/ interrupt-count = <1>;
    /inheritable/ ?interrupt-parent = phandle;
    interrupts = (cell{$(
          $interrupt-parent)/#interrupt-cells})
          {$interrupt-count};
};

{
    compatible = string("ti,dm64410-spi",
          "ti,da830-spi");

    /use/ device;
    /use/ spi {
        cs-cells = <1>;
    };
    /use/ interrupts {
        interrupt-count = <1>;
    };
    /use/ clocks {
        count = <1>;
    };

    ti,davinci-spi-intr-line = cell(<0>, <1>);
};
```

[1] [RFC 0/1] Device Tree Schema Source format proposal

```
bus: bus@0x10180000 {
    /* ... */
    #address-cells = <1>
    #interrupt-cells = <1>;
    interrupt-map-mask = <0xf800 7>;
    interrupt-map = <0x0000 1 &intc  9 3   // 1st slot
                     0x0800 1 &intc 10 3>; // 2nd slot

    device@0800 {
        /* ... */
        reg = <0x0800 0x100>;
        interrupts = <1>;
    };
};
```

Interrupt bindings.

Conclusion 2

A schema language is unlikely to cover the most complex bindings

RFC by Stephen Warren [1]

October 2013

Validation for particular bindings implemented directly in C code of dtc

Every (even trivial) binding needs to have its piece of C code

Can validate interrupt bindings!

```
void is_a_clock_consumer_by_name(struct node *node,
                                 int clock_count)
{
    required_property(node, "clock-names");
    required_property(node, "clocks");
}

static const char *compats_nvidia_tegra20_i2c[] = {
    "nvidia,tegra20-i2c",
    NULL,
};

static void checkfn_nvidia_tegra20_i2c(
                                 struct node *node)
{
    is_an_mmio_bus_child(node, 1);
    is_an_i2c_bus(node);
    is_an_interrupt_consumer_by_index(node, 1);
    is_a_clock_consumer_by_name(node, 2);
}
```

[1] [RFC PATCH dtc] C-based DT schema checker integrated into dtc

Hybrid approach to DT schema checking

Proposed solution

RFC by Tomasz Figa [1]

February 2014

C code used to validate complex generic bindings

Simple schema language used to define device-specific bindings

Best of both worlds?

```
// C file
static void generic_checkfn_interrupts(
                const struct generic_schema *schema,
                struct node *root, struct node *node,
                struct node *params, bool required)
{
    /* Interrupts validation goes here */
}
GENERIC_SCHEMA("interrupts", interrupts);

// DTSS file

/dtss-v1/;

wlf,wm8903 {
    /match/ compatible = "wlf,wm8903";

    /optional/ gpio-cfg;

    /require/ gpio-provider {
        cells = <1>;
    };

    /use/ interrupts {
        count = <1>;
    };
    // ^^ invokes generic_checkfn_interrupts()
};
```

[1] Hybrid approach for DT schema checking

Generic bindings

Generic bindings are generic

Can be used by many device bindings

Often complex

Examples:

- Busses (I2C, SPI)
- Resources (GPIO, regulators)

Implementation using C code justified

Device bindings

Bindings specific to devices

Usually built from generic bindings

Sometimes require a number of custom properties (mostly trivial)

Lots of them

Would require a patch for dtc whenever a new device is added

DTSS format

- DTS-like syntax
- One node per binding
- Binding match
- List of simple properties
- Instances of generic bindings

```
/dtss-v1/;

binding {
          /match/ match-name = ”value”

          required-property;

          /optional/ optional-property;

          /require/ required-generic-schema;

          /use/ optional-generic-schema {
                    schema-argument = <1>;
          };
};
```

Multiple matching options

compatible:
- compatible string, according to general compatible matching rules

path:
- absolute path of node

device_type:
- value of device_type property

Only one match allowed
(but might be changed?)

```
/dtss-v1/;

binding1 {
          /match/ compatible = „simple-bus”;
          /* Definition of binding 1 */
};

binding2 {
          /match/ path = „/cpus”;
          /* Definition of binding 2 */
};

binding3 {
          /match/ device_type = „memory”;
          /* Definition of binding 3 */
};
```

DTSS format – simple properties

Allows listing of simple properties

Without type checking

Can check constant values (e.g. #*-cells)

Required (by default) or optional

```
/dtss-v1/;

binding1 {
          /match/ compatible = „simple-bus”;
          /optional/ #address-cells;
          /optional/ #size-cells;
};

binding2 {
          /match/ path = „/cpus”;
          #size-cells = <0>;
};

binding3 {
          /match/ device_type = „memory”;
          reg;
};
```

DTSS format – generic bindings

Can /require/ or /use/ a generic binding

Resource-like approach

/use/ accepts missing resources

/require/ bails out on any failure

Arguments accessible by C schema implementation

```
/dtss-v1/;

binding1 {
    /match/ compatible = "wlf,wm8903”;

    /require/ gpio-provider {
        cells = <1>;
    };

    /use/ interrupts {
        count = <1>;
    };
};

binding2 {
    /match/ compatible = "nvidia,tegra20-i2c";

    /require/ i2c-bus;
};
```

Operation

1. Scan available DTSS files and build index of matching keys
2. For each node, apply all matching DTSS schemas
3. For each DTSS schema, check presence of required simple properties and execute any referenced generic schemas

Let's create a sample schema

Describes bindings shown on previous examples

```
/dtss-v1/;

arm-vic {
    /match/ compatible = "arm,pl192-vic";

    /require/ mmio-device {
        reg-count = <1>;
    };

    /require/ interrupt-controller {
        cells = <1>;
    };
};

samsung-sdhci {
    /match/ compatible = "samsung,s3c6410-sdhci";

    /require/ mmio-device {
          reg-count = <1>;
    };

    /require/ interrupts {
          count = <1>;
    };
};
```

Proposed solution

Let’s use it

```
t.figa@AMDC1227 ~/dtc $ ./dtc -O dtb -o sample.dtb sample.dts -x schema.dtss
WARNING: no schema for node /chosen
WARNING: no schema for node /aliases
WARNING: no schema for node /memory
WARNING: no schema for node /cpus
WARNING: no schema for node /cpus/cpu@0
WARNING: no schema for node /soc
ERROR: node '/soc/interrupt-controller@71200000' missing 'interrupt-controller' property
ERROR: node '/soc/interrupt-controller@71300000' missing '#interrupt-cells' property
ERROR: failed to parse interrupt entry 0 of node '/soc/sdhci@7c200000'
t.figa@AMDC1227 ~/dtc $
```

Hybrid validator at work

A (blurry) vision of the future.

What’s next?

What's next?

To do

Addressing review comments.

Specification of subnodes directly from DTSS,

Specification of simple property types from DTSS (cells, strings, phandles),

Reporting of unrecognized properties,

Sharing of parsing code with kernel,

Generating binding documentation text from DTSS,

Implementation of all the bindings.

Help needed

Device tree army needs you to:

- Review the RFC
- Share your comments
- Submit further patches
- Start using the validator

Acknowledgements

Samsung Electronics Poland
For employing me to work on Linux kernel.

Kernel Team of Samsung R&D Institute Poland
For all the time spent on discussions about Device Tree.

Linux Foundation
For organizing this conference.

Jon Smirl, Benouit Cousson, Fabien Parent, Stephen Warren
For previous attempts of Device Tree validation.

Device Tree and Linux kernel community
For all the fun of working together.

Questions?

Contact:

devicetree@vger.kernel.org
devicetree-compiler@vger.kernel.org
#devicetree at Freenode
t.figa@samsung.com

Thank you!